Studying the Impact of Policy Changes on Bug Handling Performance

Zeinab Abou Khalil

Studying the Impact of Policy Changes on Bug Handling Performance

Zeinab Abou Khalil
zeinab.aboukhalil@umons.ac.be
University of Mons, Belgium
University of Lille & Inria Lille, France

***Abstract*—The majority of the software development effort is spent on software maintenance. Bug handling constitutes one of the major software maintenance activities. Earlier studies have empirically investigated various aspects of bug handling, such as bug triaging, bug fixing, and bug process analysis. However, results from previous studies may not be applicable to contemporary agile software development practices. Moreover, these studies did not investigate how changes in the development policies and supporting tools impact the bug handling process. Therefore, our main goal is to investigate the impact of such changes on the bug handling process performance. To do so, we are conducting empirical studies on large and long-lived open source software projects. We report on our current research findings and outline the ongoing Ph.D. research project of the first author.**

I. Introduction

Continuous software engineering is a common practice for large open source software (OSS) projects that involves developing, testing, updating and deploying software releases. Every major software release introduces a significant amount of new functionality and modifies existing functionality compared to the previous release. As part of the release life cycle, the quality of each release is reviewed to ensure that it meets its specified requirements, and that the most important bugs have been resolved or fixed [1]. The software development team strives to tackle as many bugs as possible in current and upcoming releases, but in practice every deployed software release inevitably contains remaining bugs.

Many existing empirical studies have focused on understanding and improving the overall bug handling process, as well as identifying the factors affecting bug fixing and bug triaging. Very little research has focused on the distinction between what happens *before* and *after* each release, how this evolves over time, and how important changes in the development policy or in tool support impacts the performance of the bug handling process.

My PhD research project therefore proposes to empirically study these issues in large and long-lived OSS projects. In a follow-up step, the obtained research findings will be exploited to provide specific recommendations, guidelines and tools allowing both individual developers and communities to improve upon their bug handling practices.

The empirical investigation will be guided by four research questions:

RQ$_1$ ***How does the bug handling performance evolve?*** The rationale behind this question is to assess whether we can find particular trend break in the handling process over time.

RQ$_2$ ***How does the release schedule affect bug handling activity?*** This question aims to verify the assumption that the release schedule impacts the way in which bugs are handled; we expect that maintainers handle bugs more intensively in the period right before an upcoming release to increase its quality w.r.t bugs, as well as an increased bug report rate right after a new release.

RQ$_3$ ***How does the release policy influence the bug handling process?*** Some OSS projects are striving to frequently deliver updates to applications using policies such as a rolling release, rolling update, or continuous delivery [2]. A possible concern about such policies is that developers have less time to test the software and to fix all known bugs identified by developers or end users. This could imply that less bugs will be fixed before a release and that bugs may persist longer than they would for traditional release models. To assess this hypothesis, we study the characteristics of the bug handling process in pre- and post-release bug reports and compare it between previous and current adopted release policies.

RQ$_4$: *How do bug handling tools and policies affect the bug handling performance?* The rationale of this question is to investigate if introducing new policies, such as how to resolve a bug, and the use of new tools (such as bug trackers, error reporting tools and code reviewing tools) influence bug handling performance.

To answer these questions, we will follow a mixed-method research approach on a selection of OSS projects, by combining quantitative historical analyses and qualitative analyses based on interviews and online surveys with practitioners.

II. Related Work

A. Bug fixing in rapid release cycles

Khomh et al. [3] empirically studied the effect of transitioning to rapid releases on software quality for Mozilla Firefox. As quality metrics they considered runtime failures, presence of crash reports and outdatedness of used

releases. They found that less bugs are fixed during the testing period and that bugs are fixed faster in the rapid releases. In follow-up work [4] they showed that in a more rapid release model, compared to the traditional release model, less post-release bugs are fixed even though they are fixed faster. Interviewed Mozilla employees reported that they can be "less effective at triaging bugs with rapid release" and that more bugs can be generated when more beta testers use the rapid release model.

Da Costa et al. [5] studied the impact of Mozilla's rapid release cycles on the integration delay of addressed issues. They showed that addressed issues are delivered faster in the traditional releases compared to the rapid releases. They also found that the issues are triaged and fixed faster in rapid releases. In a follow-up work [6], however, they found no significant difference in triaging and fixing time between the traditional and rapid releases.

B. Bug handling characteristics

Panjer [7] studied Eclipse and found that attributes (e.g., severity, product, component, and version) of an initial bug report are the most influential factors of the bug lifetime, as well as comments in post submission information. Giger et al. [8] found that the people involved and the month when the bug was reported have the strongest influence on the bug fixing time in Mozilla, Eclipse and Gnome. Marks et al. [9] studied different features of a bug report in relation to bug fixing time in Mozilla and Eclipse projects. They found bug location and bug reporting time to be the most influential factors on bug fixing time.

Zou et al. [10] examined the characteristics of the bug fixing rate and studied the impact of a reporter's different contribution behaviors to the bug fixing rate in Eclipse and Mozilla. They observed an increase in the fixing rate over the years for both projects, however, the observed rates were not high, especially for Mozilla.

Zhang et al. [11] studied factors affecting delays incurred by developers in the bug fixing time in three Eclipse projects: Mylyn, Platform and PDE. They found that the delays in starting to address issues are due mainly to the severity, operating system, issue description and presence of comments.

Saha et al. [12] analyzed code change metrics, such as the number of changed files, to find the reasons behind bug-fixing delays and to improve the overall bug fixing process in four Eclipse projects. Their results indicate that many long-lived bugs could be reduced through careful triaging and prioritization by predicting their severity, change effort and change impact in advance.

Rwemalika [13] studied the characteristics and differences between pre-release bugs and post-release bugs in 37 industrial Java projects. They found that post-release bugs are more complex to fix since they require modification of several source code files, written in different programming languages and configuration files.

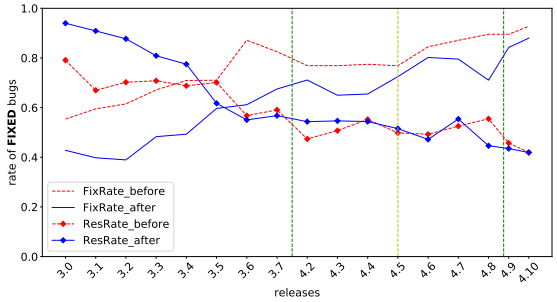

Fig. 1: Evolution of *ResRate* and *FixRate before* and *after* each release.

Zimmermann and Casanueva [14] analyzed the Coq project to determine the impact of switching the bug tracker from Bugzilla to GitHub. They found this transition to result in an increase in bug reporting, particularly from developers. They also observed an increased user engagement with the bug tracker, with more comments by developers and users.

III. Past Research

In our prior work [15], we studied the research questions through an empirical analysis of the bug handling process of four core Eclipse products over a 15-year period, considering 138K bug reports from Bugzilla, including 16 annual Eclipse releases and two quarterly releases. We relied on four metrics to quantify bug handling: bug triaging and fixing time, and bug resolution and fixing rate. Bug triaging time is the time interval between when the bug was reported and when it was assigned. Bug fixing (resp. resolution) time is the time interval between when the bug was reported and when it was fixed (resp. resolved). Bug resolution rate *ResRate* is measured as the proportion of reported bugs that have been resolved, and bug fixing rate *FixRate* is defined as the ratio of fixed over resolved bugs.

RQ*$_1$ *How does the bug handling performance evolve?

For each considered release, we computed the number of reported, resolved, fixed and assigned bugs targeting this release. We found that the number of reported bugs targeting a given release is monotonically decreasing all along the 3.x range of releases. Starting from release 4.2, the number of bug reports appears to become stable.

Based on the results of Fig. 1, we observe a *decreasing* resolution rate all along the Eclipse releases, while *FixRate* is increasing. For the 4.x annual release range we observe a stability in the rates up until release 4.5, after which *ResRate* continues to decrease and *FixRate* continues to increase. This suggests that Eclipse has a mature and high-quality bug handling process.

RQ*$_2$ *How does the release schedule affect bug handling activity?

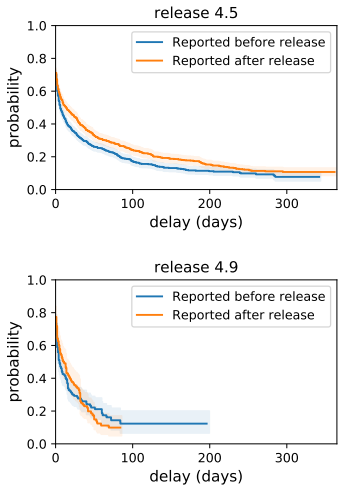

Fig. 2: Kaplan-Meier survival curve with 95% confidence interval for fixing time before and after each release.

Bug resolution and fixing rates: We computed *ResRate* and *FixRate* before and after each release date. Fig. 1 shows a decreasing trend of *ResRate* over the different releases. We did not find any significant difference between the resolution rate before and after each release. Fixing rates *before* each release were statistically higher (according to a Wilcoxon rank sum test) than fixing rates *after* the release. We also observed that *FixRate* is increasing faster since release 4.5.

Bug triaging and fixing time: We used survival analysis [16] to model the expected time duration until bug assignment and until bug fixing event occurs. As an example, Fig. 2 shows the survival curves for bug fixing before and after releases 4.5 and 4.9. Using logrank tests [17] we verified that *bug triaging time* before a release tends to be lower than triaging time after that release. We also observed that the difference between the survival curves tends to decrease over successive releases; for many recent releases the difference is very low or absent. For *bug fixing time*, we observed that it takes less time to fix a bug before compared to after a release. Starting from release 4.7 we could no longer observe statistical differences.

Bug triaging and fixing time when approaching a release deadline: For each release, we computed triaging time and fixing time for each bug. We grouped the results in two periods based on when the assignment or fixing took place: the *early period* corresponds to the first 9 months after the *current* release; and the *pressure period* corresponds to the last 3 months before the *next* release. We found that during the pressure period, bugs for the next release are handled faster than for the current release. This can be explained by an increasing deadline pressure, requiring developers to prioritise the bugs of the next release to deliver the release with fewer bugs.

For both the *current* and the *next* release, we investigate the differences in triaging and fixing time between the early period and the pressure period. For the *current* release, most analyses showed that it took longer to triage or fix bugs during the pressure period compared to the early period. These results indicate that bugs handled in the pressure period have been open for a long time and that developers tend to handle many bugs that had lived for a long time in the current release before releasing the next one. For the *next* release, we do not observe a difference between the *triaging* time of the bugs in the pressure period and early period, meaning that bugs are triaged in the same way regardless of the considered period. This means that developers try to triage these bugs as soon as possible. Moreover, there is very little difference in the time to fix bugs of the next release during the early period compared to the pressure period.

RQ$_3$ How does the release policy influence the bug handling process?

We investigated the effect of the transition from an annual to a quarterly release policy starting at release 4.9. The Eclipse community has been working towards this a transition for over a year by introducing intermediate (quarterly) "update" releases since Eclipse 4.6 in 2016. After the transition we no longer observe any difference between the number of reported, assigned and fixed bugs before and after a release, suggesting an increasingly balanced bug handling workload. Also, after the transition we no longer observe any statistical difference in triaging and fixing time before and after releases (see Fig. 2). Our results highlight the importance of well preparing the transition from a traditional to a rapid release policy so as the community to become more effective in bug handling activities.

RQ$_4$ How do tools and policies affect the bug handling performance?

Error reporting tool: Eclipse 4.5 introduced the Automated Error Reporting client (AERI). The Eclipse community considered AERI as beneficial: it eases reporting errors as users do not need to create Bugzilla entries; they report the issue directly from within Eclipse. In turn, users can provide comments with their reports which are helpful when fixing bugs; according to [18] commented bug reports are fixed twice as fast.Our empirical analysis confirmed AERI's positive effect on the bug fixing rate. Fig. 1 shows that *FixRate* before and after a release is improving faster since release 4.5. We assume that this change is caused by the introduction and continuous use of AERI.

Change of resolution policy: In 2007 Eclipse decided to stop using the resolution statuses LATER and REMIND as they gave rise to bugs that remained unresolved for too long. By analyzing the delays of a follow-up resolution of the REMIND/LATER bugs (3479 bugs), we found that most of them lingered for more than 3 years before getting their follow-up resolution. The decision to stop using these resolution statuses coincided with a decrease in resolution rate, while the fixing rate increased.

IV. Ongoing Research

This section summarizes our ongoing work, where we aim to: (i) generalise our results, (ii) perform more quantitative analyses, (iii) conduct qualitative analyses to confirm our results, and (iv) study the impact of other important tool and policy changes.

Result generalisation: More systems that have changed their release policy will be investigated to uncover factors affecting the bug handling process w.r.t. our RQs. For instance, Mozilla Firefox has shifted from the traditional development model to a rapid release model. All our research questions will be studied for Firefox to analyze its bug handling performance over releases, taking into account specific policy or tool changes. We will also compare the effect of switching to a rapid release policy between Eclipse and Firefox.

The traditional release cycle of Firefox was applied to major releases (1.0 to 4.0) that would take 12-18 months to be shipped. Firefox adopted a rapid release cycle in March 2011 where a major release is shipped every 6 weeks. We retrieved the bug history of all actual bugs of Firefox, by excluding enhancements, reported between Firefox 3.5 and Firefox 60, resulting in a dataset with 221,048 bugs. Since approximately 80% of the bugs are missing their version field, we will link each bug to a version using Version Control System (VCS) data and by following the approach of Da Costa et al. [5]. This is achieved by extracting the commit logs and finding the commits fixing specific issues (as their ID is mentioned in the commit message). Bugzilla also contains tracking flag information that can assist in linking bugs to releases; flags are used by developers, triagers, QA and the release management teams to keep track of bugs that affect particular release(s).

More quantitative analyses: We will use regression discontinuity analysis to confirm the causal of policy changes [19]. We will use multiple regression models to study which combination of factors impacts the bug handling process. We will also study if the type (e.g security related) and severity of the bugs that are triaged and fixed faster in the new adopted release policy are different from the previous one. Such analyses will disclose how practices are affected by changes in the release policy.

Qualitative analyses: As quantitative analyses provide limited insights, we also plan to perform qualitative analyses since they are appropriate to retrieve the pragmatic stance of software engineering research [20]. We will conduct semi-structured interviews with the developer communities of the selected projects. We are currently studying the bug handling performance of bug reports from two systems and we are planning to interview developers to find out the preparations that took place when changing policy and bug handling practices. The goal of these interviews is to better explain our results and find out if the selected bug handling process changed after changing release policy. Moreover, the difficulties faced when changing the release policy will be discussed during the interviews to gain developers' perspective of the possible impact on the bug handling process and what aspects of bug handling performance developers care about.

Impact of switching to new tools: For most software projects, bug tracking tools are vital for managing bugs. Broadly, in such tools, all users have access to bug reporting. Given the impact they can have on the bug handling process, it is essential to study the performance of a community when using different bug tracking environments.

To the best of our knowledge, only Zimmermann and Casanueva [14] have measured the impact of changing the bug reporting environment when Coq, an open source proof assistant, switched from Bugzilla to GitHub. They found increased developer activity when reporting their own bugs and discussing bug reports on GitHub. Moreover, the users were more likely to have an active role by commenting on bug reports. This is an indicator of shift in the dynamics of the bug reporting process as more people are engaged in the process, and openness and transparency are increased. Zimmerman and Casanueva however, only considered two outcome variables: the number of bug reports and comments. I aim to analyze other variables such as the speed of bug resolution, bug tossing, resolution rate and bug reopening rate.For instance, the OpenStack Infrastructure team has migrated all of their project bugs from LaunchPad to StoryBoard [21].

Process mining: As a complementary way to study the effect of switching to a new tool or policy, we will use the technique of process mining [22]. We plan to use this technique to find if and how the bug handling process has changed after the switch.

V. Conclusion

This paper reports on our current research findings and outlines the proposed research work of the first author toward earning her PhD. This work is motivated by the need to understand the evolution of the bug handling performance in OSS and how changes in development policies and supporting tools can impact the bug handling process. Our main goal is to investigate the impact of such changes on the bug handling process performance and in a follow-up step, try to improve it. To do so, we are currently conducting empirical studies on large and long-lived open source software projects.

Acknowledgment

This paper reports on the ongoing PhD research by the first author, under joint supervision by Pr. Tom Mens and Eleni Constantinou (University of Mons) and Pr. Laurence Duchien and Clément Quinton (University of Lille). This research was partially supported by the FNRS and FWO under the Excellence of Science project 30446992 SECO-ASSIST and the FRQ-FNRS collaborative research project R.60.04.18.F SECO-Health. Zeinab Abou Khalil is partially supported by Région Hauts-de-France.

References

[1] A. Mockus, R. T. Fielding, and J. D. Herbsleb, "Two case studies of open source software development: Apache and mozilla," *ACM Transactions on Software Engineering and Methodology*, vol. 11, pp. 309–346, July 2002.

[2] Wikipedia contributors, "Rolling release." https://en.wikipedia.org/w/index.php?title=Rolling_release&oldid=896696167.

[3] F. Khomh, T. Dhaliwal, Y. Zou, and B. Adams, "Do faster releases improve software quality? An empirical case study of Mozilla Firefox," in *IEEE Working Conf. Mining Software Repositories (MSR)*, pp. 179–188, June 2012.

[4] F. Khomh, B. Adams, T. Dhaliwal, and Y. Zou, "Understanding the impact of rapid releases on software quality," *Empirical Software Engineering*, vol. 20, pp. 336–373, Apr 2015.

[5] D. A. da Costa, S. McIntosh, U. Kulesza, and A. E. Hassan, "The impact of switching to a rapid release cycle on the integration delay of addressed issues - An empirical study of the Mozilla Firefox project," in *Working Conference on Mining Software Repositories (MSR)*, pp. 374–385, IEEE, 2016.

[6] D. A. da Costa, S. McIntosh, C. Treude, U. Kulesza, and A. E. Hassan, "The impact of rapid release cycles on the integration delay of fixed issues," *Empirical Software Engineering*, pp. 1–70, 2018.

[7] L. D. Panjer, "Predicting Eclipse bug lifetimes," in *International Workshop on Mining Software Repositories*, p. 29, IEEE Computer Society, 2007.

[8] E. Giger, M. Pinzger, and H. Gall, "Predicting the fix time of bugs," in *International Workshop on Recommendation Systems for Software Engineering*, pp. 52–56, ACM, 2010.

[9] L. Marks, Y. Zou, and A. E. Hassan, "Studying the fix-time for bugs in large open source projects," in *International Conference on Predictive Models in Software Engineering*, ACM, 2011.

[10] W. Zou, X. Xia, W. Zhang, Z. Chen, and D. Lo, "An empirical study of bug fixing rate," in *Computer Software and Applications Conference (COMPSAC)*, pp. 254–263, IEEE, 2015.

[11] F. Zhang, F. Khomh, Y. Zou, and A. E. Hassan, "An empirical study on factors impacting bug fixing time," in *Working Conference on Reverse Engineering*, pp. 225–234, IEEE, 2012.

[12] R. K. Saha, S. Khurshid, and D. E. Perry, "Understanding the triaging and fixing processes of long lived bugs," *Information and software technology*, vol. 65, pp. 114–128, 2015.

[13] R. Rwemalika, M. Kintis, M. Papadakis, Y. Le Traon, and P. Lorrach, "An industrial study on the differences between pre-release and post-release bugs," in *2019 35th International Conference on Software Maintenance and Evolution (ICSME)*, pp. 1–12, IEEE, 2019.

[14] T. Zimmermann and A. Casanueva Artís, "Impact of switching bug trackers: a case study on a medium-sized open source project," in *2019 35th International Conference on Software Maintenance and Evolution (ICSME)*, pp. 1–12, IEEE, Mar. 2019.

[15] Z. Abou Khalil, E. Constantinou, T. Mens, L. Duchien, and C. Quinton, "A longitudinal analysis of bug handling across Eclipse releases," in *2019 35th International Conference on Software Maintenance and Evolution (ICSME)*, pp. 1–12, IEEE, 2019.

[16] O. Aalen, O. Borgan, and H. Gjessing, *Survival and event history analysis: a process point of view*. Springer Science & Business Media, 2008.

[17] J. M. Bland and D. G. Altman, "The logrank test," *BMJ*, vol. 328, no. 7447, p. 1073, 2004.

[18] A. Sewe, "One year of automated error reporting." https://www.eclipse.org/community/eclipse_newsletter/2016/july/article3.php, July 2016.

[19] J. D. Angrist and J.-S. Pischke, *Mastering'metrics: The path from cause to effect*. Princeton University Press, 2014.

[20] S. Easterbrook, J. Singer, M.-A. Storey, and D. Damian, *Selecting Empirical Methods for Software Engineering Research*, pp. 285–311. London: Springer London, 2008.

[21] Michael Krotscheck, "Goodbye launchpad, hello storyboard." https://krotscheck.net/2014/11/20/goodbye-launchpad-hello-storyboard.html, 2014.

[22] W. Poncin, A. Serebrenik, and M. Van Den Brand, "Process mining software repositories," in *2011 15th European Conference on Software Maintenance and Reengineering*, pp. 5–14, IEEE, 2011.